

appo

应用编排器appo是负责编排应用生命周期维护操作的核心模块。

v1.2版本说明 http://docs.edgegallery.org/zh_CN/release-v1.2/Projects/MECM/MECM_Architecture_en.html#application-orchestrator

接口设计

创建实例

入口代码

业务流程

Create app instance

Get configuration from inventory

DownloadAndDecompose

总结

实例化实例

入口代码

业务流程

Start create app instance

Get configuration from inventory

Instantiate

接口设计

appo主要涉及以下接口：

Appo Interfaces

- [Get All Application Instances](#)
- [Get Application Instance](#)
- [Create AppInstance](#) //创建app实例
- [Get App Instance Information](#)
- [Instantiate Application](#) //app实例化
- [Delete Application Instance](#)
- [Get Query Kpi](#)
- [Get Mep Capabilities](#)
- [Get Mep Capability](#)
- [Create Batch](#)
- [Batch instantiate](#) //批量实例化
- [Batch terminate](#)
- [Batch Query](#)
- [Queries liveness & readiness](#)
- [Create AppRule Config](#) //创建appRule config
- [Update AppRule Config](#)
- [Delete AppRule Config](#)
- [Get AppRule Config Status](#)
- [Synchronizes application instance info from edge](#)
- [APPO Open Api Swagger](#)

下面以创建实例和实例化实例两个接口来进行分析：

创建实例

入口代码

创建实例的接口为 /appo/v1/tenants/{tenant_id}/app_instances, 实现入口位于mecm-appo/src/main/java/org/edgegallery/mecm/appo/apihandler/AppOrchestratorHandler.java:

```
1  @ApiOperation(value = "Creates application instance", response = AppoResponse.class)
2  @PostMapping(path = "/tenants/{tenant_id}/app_instances", produces = MediaType.APPLICATION_JSON_VALUE)
3  @ApiResponses(value = {@ApiResponse(code = 201, message = "request accepted ", response = AppoResponse.class),
4  @ApiResponse(code = 500, message = "internal server error", response = String.class)
5  })
6  @PreAuthorize("hasRole('MECM_TENANT') || hasRole('MECM_ADMIN')")
7  public ResponseEntity<AppoResponse> createAppInstance(
8  @ApiParam(value = "access token") @RequestHeader("access_token") String accessToken,
9  @ApiParam(value = "tenant id") @PathVariable("tenant_id")
10 @Pattern(regexp = Constants.TENENT_ID_REGEX) @Size(max = 64) String tenantId,
11 @ApiParam(value = "create application instance")
12 @Valid @RequestBody CreateParam createParam) {
13     logger.debug("Application create request received...");
14     //createParam除了包含appPackage,appName,appId,mep服务等信息, 还包含了mecHost, 即边缘节点ip
15     return appoService.createAppInstance(accessToken, tenantId, createParam);
16 }
```

其中createParam的结构如下:

```
1  public abstract class AppInstanceParam {
2
3  @NotEmpty(message = "Package ID is mandatory")
4  @Size(max = 64)
5  @Pattern(regexp = APP_PKG_ID_REGEX, message = "Package ID is invalid. It must be lowercase letters or digits with "
6  + "length of 32/64 characters.")
7  private String appPackageId;
8
9  @NotEmpty(message = "Application name is mandatory")
10 @Size(max = 128)
11 @Pattern(regexp = APP_NAME_REGEX, message = "App name is invalid. It must start and end with alpha numeric "
12 + "character and special characters allowed are hyphen and underscore.")
13 private String appName;
14
15 @NotEmpty(message = "Application instance ID is mandatory")
16 @Size(max = 64)
17 @Pattern(regexp = APPD_ID_REGEX, message = "Application instance ID is invalid. It must be lowercase letters or "
18 + "digits with length of 32 characters.")
19 private String appId;
20
21 @NotEmpty(message = "Application instance description is mandatory")
22 @Size(max = 256)
23 private String appInstanceDescription;
24
25 @Size(max = 10, message = "capabilities exceeds max limit 10")
26 private List<@Valid String> hwCapabilities = new LinkedList<>();
27 }
28 public final class CreateParam extends AppInstanceParam {
29
30 @NotEmpty(message = "MEC host is mandatory")
31 @Size(max = 15)
32 @Pattern(regexp = HOST_IP_REGEX, message = "MEC host IP is invalid")
33 private String mecHost;
34 }
```

继续看createAppInstance的实现:

```
1  @Override
2  public ResponseEntity<AppoResponse> createAppInstance(String accessToken, String tenantId,
3  CreateParam createParam) {
4  LOGGER.debug("Application create request received...");
```

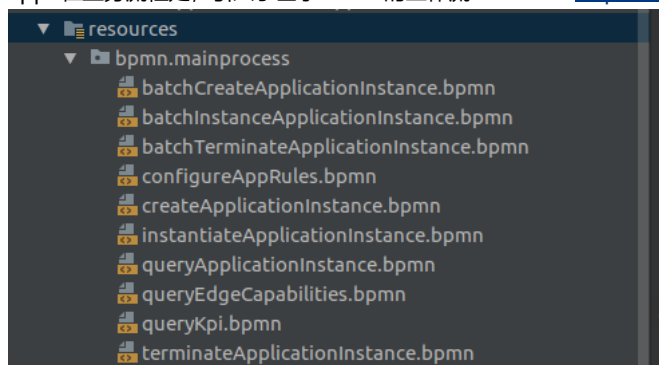
```

5 //re-use check,同租户下,不能在同一mecHost上创建同名app
6 //...省略
7 //创建工作流入参
8 Map<String, String> requestBodyParam = new HashMap<>();
9 requestBodyParam.put(Constants.TENANT_ID, tenantId);
10 requestBodyParam.put(Constants.APP_PACKAGE_ID, createParam.getAppPackageId());
11 requestBodyParam.put(Constants.APP_ID, createParam.getAppId());
12 requestBodyParam.put(Constants.APP_NAME, createParam.getAppName());
13 requestBodyParam.put(Constants.APP_DESCR, createParam.getAppInstanceDescription());
14 requestBodyParam.put(Constants.MEC_HOST, createParam.getMecHost());
15 String hwCapabilities =
16 createParam.getHwCapabilities().stream().map(Object::toString).collect(Collectors.joining(", "));
17 requestBodyParam.put(Constants.HW_CAPABILITIES, hwCapabilities);
18
19 LOGGER.debug("Create instance input parameters: {}", requestBodyParam);
20 String appInstanceId = UUID.randomUUID().toString();
21 requestBodyParam.put(Constants.APP_INSTANCE_ID, appInstanceId);
22 requestBodyParam.put(Constants.ACCESS_TOKEN, accessToken);
23 //写DB, 创建app instance记录
24 AppInstanceInfo appInstInfo = new AppInstanceInfo();
25 appInstInfo.setAppInstanceId(appInstanceId);
26 appInstInfo.setAppPackageId(createParam.getAppPackageId());
27 appInstInfo.setTenant(tenantId);
28 appInstInfo.setAppId(createParam.getAppId());
29 appInstInfo.setAppName(createParam getAppName());
30 appInstInfo.setAppDescriptor(createParam.getAppInstanceDescription());
31 appInstInfo.setMecHost(createParam.getMecHost());
32 appInstInfo.setOperationalStatus(Constants.OPER_STATUS_CREATING);
33 appInstanceInfoService.createAppInstanceInfo(tenantId, appInstInfo);
34
35 requestBodyParam.put(Constants.APPRULE_TASK_ID, appInstanceId);
36 //写DB, 给app instance记录创建关联的app rule
37 AppRuleTask appRuleTaskInfo = new AppRuleTask();
38 appRuleTaskInfo.setAppRuleTaskId(appInstanceId);
39 appRuleTaskInfo.setTenant(tenantId);
40 appRuleTaskInfo.setAppInstanceId(appInstanceId);
41 appRuleTaskInfo.setConfigResult(APP_RULE_PROCESSING);
42 appInstanceInfoService.createAppRuleTaskInfo(tenantId, appRuleTaskInfo);
43 //执行BPMN workflow, 流程名称为createApplicationInstance, 流程入参requestBodyParam
44 processflowService.executeProcessAsync("createApplicationInstance", requestBodyParam);
45 Map<String, String> response = new HashMap<>();
46 response.put(Constants.APP_INSTANCE_ID, appInstanceId);
47 return new ResponseEntity<>(new AppoResponse(response), HttpStatus.ACCEPTED);
48 }

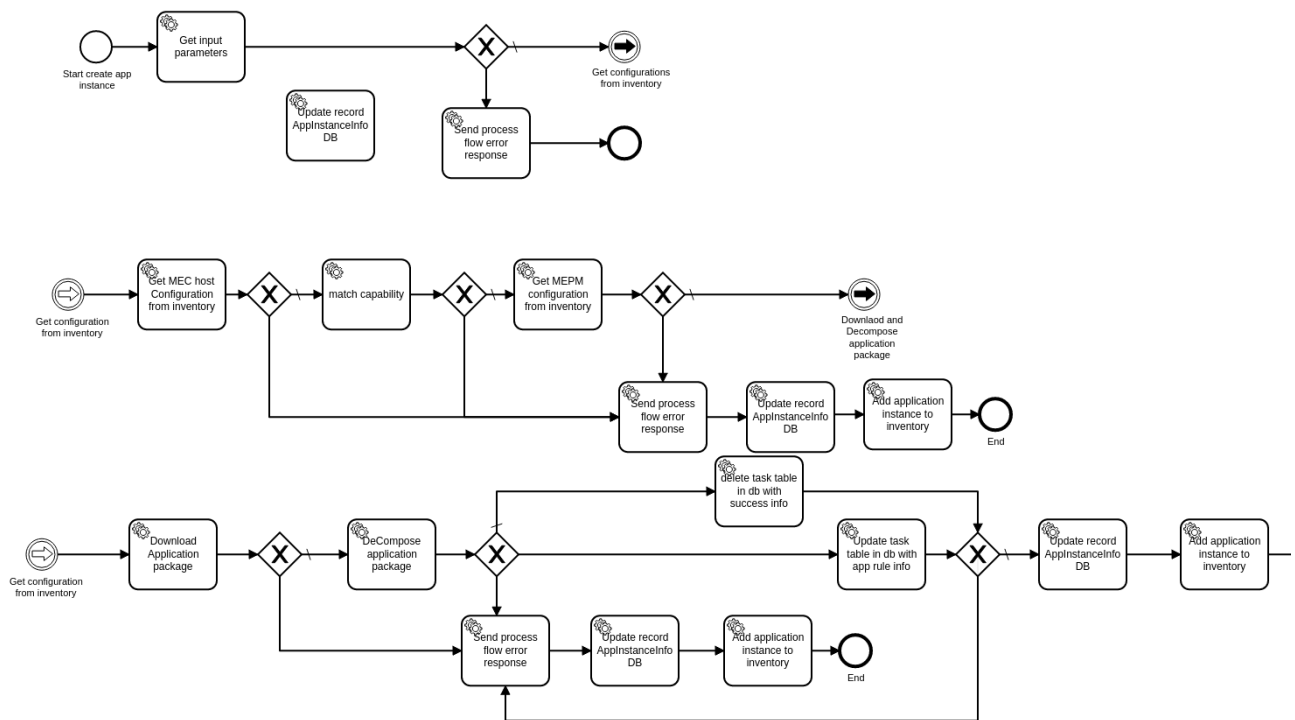
```

业务流程

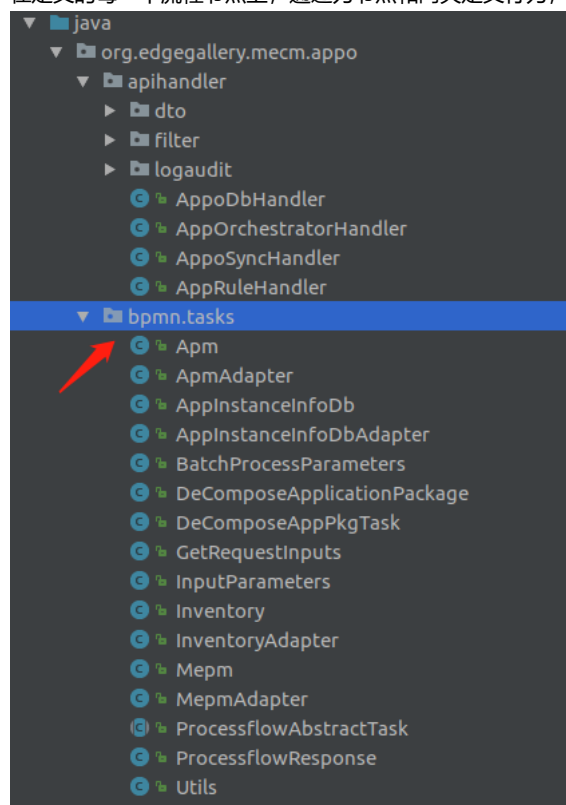
appo在业务流程处, 引入了基于BPMN的工作流Camunda <https://camunda.com/>, 其流程定义位于:



在安装完Camunda插件后, 打开createApplicationInstance.bpmn插件, 可查看其流程图:



在定义的每一个流程节点上，通过为节点和网关定义行为，执行具体的业务代码，业务代码位于：



创建app处具体包括3个流程，分别为：

1. Create app instance
2. Get configuration from inventory
3. DownloadAndDecompose （注，这里不知道是不是流程名写错了）

格式：

[节点

代码入口，节点入参，业务逻辑]

Create app instance

1. Get input parameter

bpmn.tasks.GetRequestInputs requestAction=CreateAppInstance 向DelegateExecution赋值各种AppInstance信息, [DelegateExecution](#)即为流程上下文, 如果出错, 执行错误流程

2. Exclusive 网关

无 无 执行节点网关, 如果出错, 执行错误流程bpmn.tasks.ProcessflowResponse, 返回失败。否则, 执行成功的话link到inventory流程

错误处理流程:

1. Send process flow error response

bpmn.tasks.ProcessflowResponse responseType=failure 从上下文中解析reponse,如果失败, 将流程code和resp写入上下文, 打印日志

Get configuration from inventory

1. Get MEC host Configuration from inventory

bpmn.tasks.InventoryAdapter inventory=mecHost 根据DelegateExecution中的mecHost ID,调用inventory服务获取mecHost上的信息 (包括applcm_ip, app_rule_manager_ip, mec能力列表hw_capabilities_list) , 并写入上下文, 如果成功, 执行match capability, 错误执行错误流程。

2. match capability

bpmn.tasks.Utils utilType=matchCapabilities 对app的需求能力列表和mec host提供的能力列表做比对。失败进入错误流程, 成功继续

3. Get MEPM configuration from inventory

tasks.InventoryAdapter inventory=mepm 根据入参, 执行getMepm函数, 具体为根据上下文中的mepm_ip和tenant_id, 调用[inventory接口/inventory/v1/mepms/{mepm_ip}](#), 获取mepmPort后写入上下文, 成功后link到DownloadAndDecompose, 错误执行错误流程。

错误处理流程:

1. Send process flow error response

bpmn.tasks.ProcessflowResponse responseType=failure 向上下文写入错误码和resp

2. Update record AppInstanceInfo DB

bpmn.tasks.AppInstanceInfoDbAdapter operationType=update, operational_status=Create failed 通过上下文从DB中获取appInstance记录后, 更新记录状态

3. Add application instance to inventory

bpmn.tasks.InventoryAdapter inventory=application,operType=ADD,status=Create failed 从上下文读值mecHost、tenantId、appInstanceId、appName和appPackageId,构造请求, 调用inventory接口[POST /inventory/v1/tenants/{tenant_id}/mechosts/{mec_host}/apps](#), 其中status为Create failed

DownloadAndDecompose

1. Download Application package

tasks.ApmAdapter operationType=download 从上下文获取appPkgId,调用apm服务接口 GET /apm/v1/tenants/{tenant_id}/packages/{app_package_id}/download , 返回app包的文件对象, 然后copy到以appInstanceId为粒度的指定目录下{base}/{appInstanceId}/xxx, 成功继续, 失败进入错误处理流程。

2. DeCompose application package

tasks.DeComposeApplicationPackage 无参数 解压appPkg包, 解析TOSCA.meta, 从properties字段获取app_rules后写入上下文。失败进入错误处理流程, 成功继续。

接下来根据appRules的值, 从任务delete task table in db with success info和Update task table in db with app rule info中选其一执行。当上下文中app_rules的值不为空时:

3.1 Update task table in db with app rule info

tasks.AppInstanceInfoDbAdapter operationType=updateAppRuleTask, ResponseCode=null 获取DelegateExecution中的app_rules, 更新DB的appRuleTask, 失败转错误处理, 成功继续

当DelegateExecution对象中app_rules的值为空时:

3.2 delete task table in db with success info

tasks.ApplInstanceInfoDbAdapter operationType=deleteAppRuleTask 根据租户id和app id, 从DB上删除appRuleTask信息。

4. Update record ApplInstanceInfo DB

tasks.ApplInstanceInfoDbAdapter operationType=update, operational_status=Created 将DB中的applInstance记录状态更新为Created

5. Add application instance to inventory

tasks.InventoryAdapter inventory=application, operType=ADD, status=Created 调用inventory接口 POST /inventory/v1/tenants/{tenant_id}/mechosts/{mec_host}/apps,想inventory中添加实例信息

6. Send process flow error response

tasks.ProcessflowResponse responseTpye=success 执行success分支, 向上下文添加resp和resp code

错误处理流程:

1. Send process flow error response

bpmn.tasks.ProcessflowResponse responseType=failure 向上下文写入错误码和resp

2. Update record ApplInstanceInfo DB

bpmn.tasks.ApplInstanceInfoDbAdapter operationType=update, operational_status=Create failed 通过上下文从DB中获取applInstance记录后, 更新记录状态

3. Add application instance to inventory

bpmn.tasks.InventoryAdapter inventory=application,operType=ADD,status=Create failed 从上下文读值mecHost、tenantId、applInstanceid、appName和appPackageId,构造请求, 调用inventory接口POST /inventory/v1/tenants/{tenant_id}/mechosts/{mec_host}/apps, 其中status为Create failed

总结

CreateApplicationInstance整体的过程其实是一个写DB的过程, 通过入参, 解析出package等信息, 通过和inventory交互, 做一些依赖检查, 更新自身的applInstance和appRule记录, 并最终向inventory推送。

实例化实例

入口代码

app实例化的接口定义位于

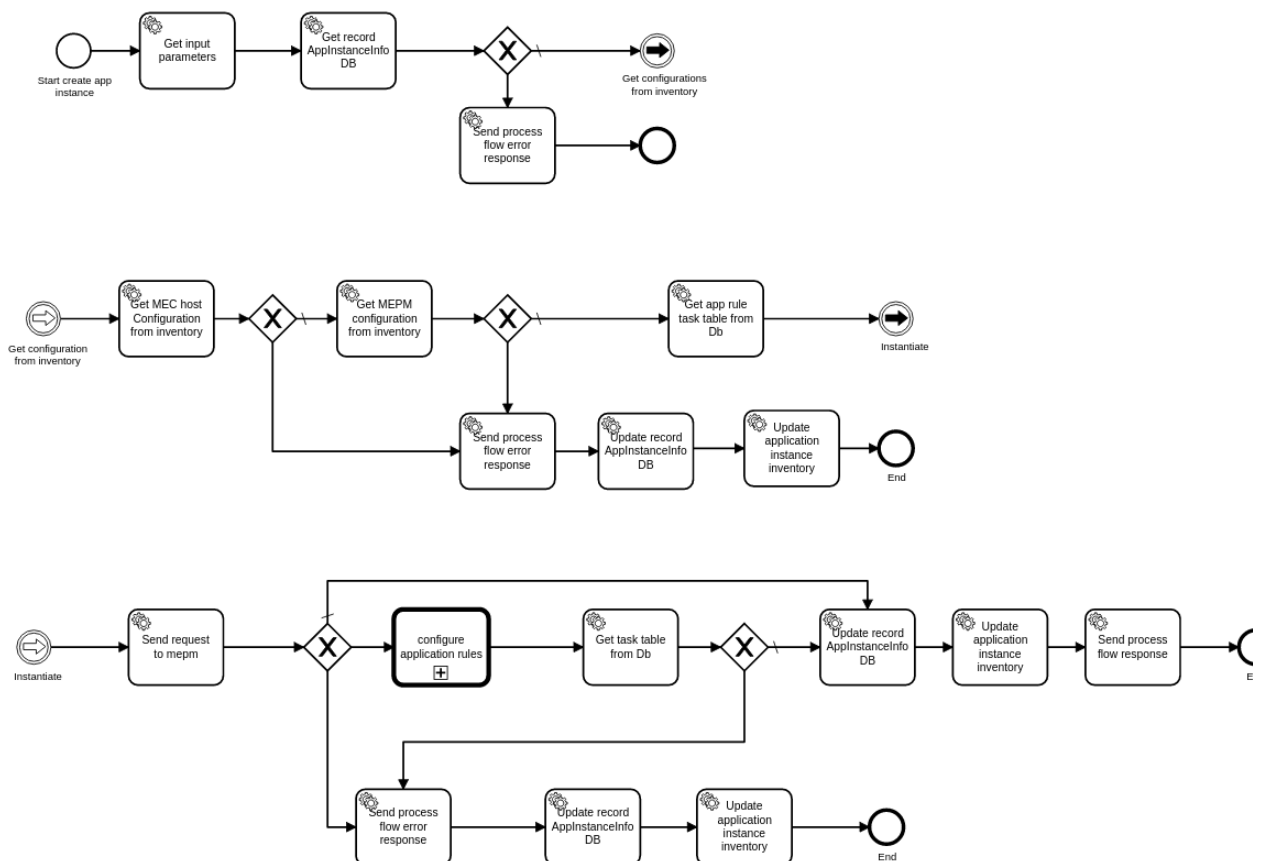
```
1
2 @ApiOperation(value = "Instantiate application instance", response = AppoResponse.class)
3 @PostMapping(path = "/tenants/{tenant_id}/app_instances/{app_instance_id}",
4 produces = MediaType.APPLICATION_JSON_VALUE)
5 @ApiResponses(value = {@ApiResponse(code = 201, message = "request accepted ", response = AppoResponse.class),
6 @ApiResponse(code = 500, message = "internal server error", response = String.class)
7 })
8 @PreAuthorize("hasRole('MECM_TENANT') || hasRole('MECM_ADMIN')")
9 public ResponseEntity<AppoResponse> instantiateAppInstance(
10 @ApiParam(value = "access token") @RequestHeader("access_token") String accessToken,
11 @ApiParam(value = "tenant id") @PathVariable("tenant_id")
12 @Pattern(regexp = Constants.TENENT_ID_REGEX) @Size(max = 64) String tenantId,
13 @ApiParam(value = "application instance id")
14 @PathVariable("app_instance_id") @Pattern(regexp = Constants.APP_INST_ID_REGEX)
15 @Size(max = 64) String appInstanceId,
16 @ApiParam(value = "Instantiate application instances")
17 @Valid @RequestBody(required = false) AppInstantiateReqParam instantiateParam) {
18
19 logger.debug("Application instantiation request received...");
20
21 return appoService.instantiateAppInstance(accessToken, tenantId, appInstanceId, instantiateParam);
22 }
```

函数实现上主要就是根据DB中的内容，启动instantiateApplicationInstance流程：

```
1 @Override
2 public ResponseEntity<AppoResponse> instantiateAppInstance(String accessToken, String tenantId,
3 String appInstanceId,
4 AppInstantiateReqParam instantiationParams) {
5     LOGGER.debug("Application instantiation request received...");
6     AppInstanceInfo appInstanceInfo = appInstanceInfoService.getAppInstanceInfo(tenantId, appInstanceId);
7     String operationalStatus = appInstanceInfo.getOperationalStatus();
8     //...param check
9     Map<String, String> requestBodyParam = new HashMap<>();
10    requestBodyParam.put(Constants.TENANT_ID, tenantId);
11    requestBodyParam.put(Constants.APP_INSTANCE_ID, appInstanceId);
12    LOGGER.debug("Instantiate input params: {}", requestBodyParam);
13    requestBodyParam.put(Constants.APPRULE_TASK_ID, appInstanceId);
14    requestBodyParam.put(Constants.ACCESS_TOKEN, accessToken);
15    if (instantiationParams != null && !instantiationParams.getParameters().isEmpty()) {
16        requestBodyParam.put(Constants.INSTANTIATION_PARAMS,
17            new Gson().toJson(instantiationParams.getParameters()));
18    }
19    //...启动流程instantiateApplicationInstance
20    processflowService.executeProcessAsync("instantiateApplicationInstance", requestBodyParam);
21    return new ResponseEntity<>(new AppoResponse(HttpStatus.ACCEPTED), HttpStatus.ACCEPTED);
22 }
```

业务流程

instantiateApplicationInstance的整体过程与上一节类似，流程图如下：



主要涉及三个流程：

1. Start create app instance
2. Get configuration from inventory
3. Instantiate

Start create app instance

1. Get input parameters

tasks.GetRequestInputs requestAction=InstantiateAppInstance 给流程上下文赋值accessToken, tenantId, appInstanceId, appRuleTaskId

2. Get record AppInstanceInfo DB

tasks.AppInstanceInfoDbAdapter operationType=get 根据tenantId和appInstanceId获取appInstance的db记录, 赋值给上下文

错误处理流程:

1. Send process flow error response

tasks.ProcessflowResponse responseType=failure 向上下文赋值错误码和错误信息

Get configuration from inventory

1. Get MEC host Configuration from inventory

tasks.InventoryAdapter inventory=mecHost 根据mecHost和tenantId, 调inventory接口GET /inventory/v1/mechosts/{mec_host}, 得到mepmlp以及mecHost上的能力列表, 赋值给上下文

2. Get MEPM configuration from inventory

tasks.InventoryAdapter inventory=mepm 根据mepmlp和tenantId, 调inventory接口GET /inventory/v1/mepms/{mepm_ip}, 获取mepm的port, 写入上下文

3. Get app rule task table from Db

tasks.AppInstanceInfoDbAdapter operationType=getAppRuleTask 从上下文获取appRuleTaskId和tenantId, 查db得到appRuleTaskInfo, 赋值流程app_rules和app_rule_status

错误处理流程:

1. Send process flow error response

tasks.ProcessflowResponse responseType=failure 向上下文赋值错误码和错误信息

2. Update record AppInstanceInfo DB

tasks.AppInstanceInfoDbAdapter operationType=update,operational_status=Instantiation failed 从上下文中取出appInstanceInfo信息, 写入error状态字段, 更新DB

3. Update application instance inventory

tasks.InventoryAdapter inventory=application,operType=UPDATE,status=Instantiation failed 调inventory接口GET /inventory/v1/tenants/{tenant_id}/mechosts/{mec_host}/apps/{app_instance_id}获取appInstance在inventory中的记录, 再调用inventory接口PUT /inventory/v1/tenants/{tenant_id}/mechosts/{mec_host}/apps/{app_instance_id},更新DB status字段

Instantiate

1. Send request to mepm

tasks.MepmAdapter action=instantiate 从上下文获取appInstanceInfo和实例化参数appInstantiationParams, 封装appInstReq, 其中包括了mecHostIp, packageId, appName和初始化参数。调用lcm接口 POST {mepmlp:port}/lcmcontroller/v1/tenants/{tenant_id}/app_instances/{app_instance_id}/instantiate,如果成功调用, 删除本地的appPkg包

如果上下文的app_rules不为空, 即在网关后的流程线上有如下判断:

SequenceFlow_1sy27gr

General Listeners Extensions

General

Id
SequenceFlow_1sy27gr

Name

Details

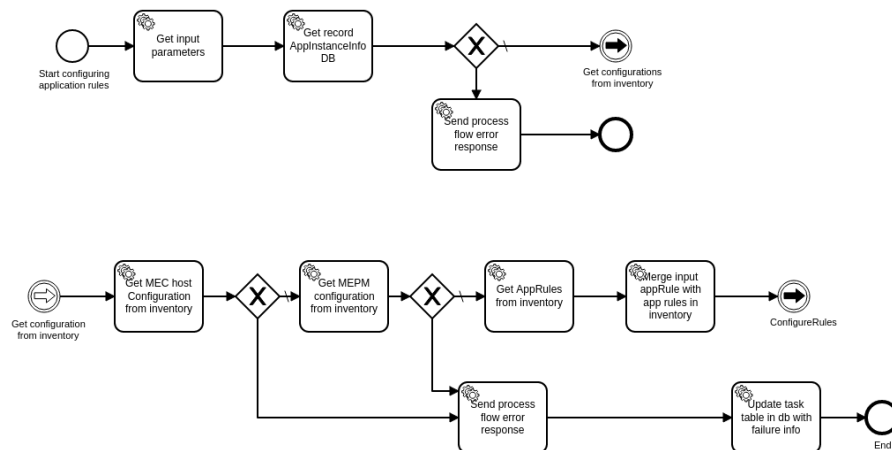
Condition Type
Expression

Expression
#{execution.getVariable("app_rules")==null}

Documentation

2.1. configure application rules

configureAppRules app_rule_action=POST 此处调用了一个子流程configureAppRules:



其初始化的上下文赋值配置如下:

Variables

In Mapping

```

access_token := access_token
tenant_id := tenant_id
app_package_id := app_package_id
app_instance_description := app_instance_description
app_id := app_id
app_name := app_name
app_instance_id := app_instance_id
hw_capabilities := hw_capabilities
mec_host := mec_host
apprule_task_id := app_instance_id
app_rules := app_rules
  
```

子流程的具体工作为:

1. Get input parameters tasks.GetRequestInputs requestAction=ConfigureAppRule 向流程上下文赋值各种参数, 比如 appInstanceId tenantId等
2. Get record AppInstanceInfo DB tasks.AppInstanceInfoDbAdapter operationType=get 调用DB, 获取appInstanceInfo 并赋值给上下文
3. Get MEC host Configuration from inventory tasks.InventoryAdapter inventory=mecHost 取出上下文中的 mecHost字段, 调用inventory接口GET /inventory/v1/mechosts/{mec_host}获取mepmIp和mecHost上的mep能力, 赋值上下文
4. Get MEPM configuration from inventory tasks.InventoryAdapter inventory=mepm 从上下文取出mepmIp,调用 inventory GET /inventory/v1/mepms/{mepm_ip}接口得到mepmPort, 赋值上下文
5. Get AppRules from inventory tasks.InventoryAdapter inventory=apprules,operType=GET 从上下文获取 appInstanceId 调用inventory GET /inventory/v1/tenants/{tenant_id}/app_instances/{app_instance_id}/appd_configuration, 将返回赋值给上下文 inventory_app_rules
6. Merge input appRule with app rules in inventory tasks.Utils utilType=mergeAppRules 从上下文分别获取 appRules和inventoryAppRules, 主要逻辑为合并两类rule, 当rule有重合, 使用inventoryAppRules, 当appRules中没有 inventoryAppRules, 则向appRules增加。并将最后的结果添加入上下文。

2.2. Get task table from Db

tasks.AppInstanceInfoDbAdapter operationType=getAppRuleTask 从上下文获取appRuleTaskId和tenantId, 查DB得到appRuleInfo,信息写入DB

3.Update record AppInstanceInfo DB

tasks.AppInstanceInfoDbAdapter operationType=update, operational_status=Instantiated 从上下文中取出appInstanceInfo信息, 写入Instantiated状态字段, 更新DB

4.Update application instance inventory

tasks.InventoryAdapter inventory=application, operType=UPDATE, status=instantiated, 调inventory接口GET /inventory/v1/tenants/{tenant_id}/mechosts/{mec_host}/apps/{app_instance_id}获取appInstance在inventory中的记录, 再调用inventory接口PUT /inventory/v1/tenants/{tenant_id}/mechosts/{mec_host}/apps/{app_instance_id},更新DB status字段为instantiated

5. Send process flow response

tasks.ProcessflowResponse responseType=success 上下文写入success记录

错误处理流程:

1. Send process flow error response

tasks.ProcessflowResponse responseType=failure 向上下文赋值错误码和错误信息

2. Update record AppInstanceInfo DB

tasks.AppInstanceInfoDbAdapter operationType=update,operational_status=Instantiation failed 从上下文中取出appInstanceInfo信息, 写入error状态字段, 更新DB

3. Update application instance inventory

tasks.InventoryAdapter inventory=application,operType=UPDATE,status=Instantiation failed 调inventory接口GET /inventory/v1/tenants/{tenant_id}/mechosts/{mec_host}/apps/{app_instance_id}获取appInstance在inventory中的记录, 再调用inventory接口PUT /inventory/v1/tenants/{tenant_id}/mechosts/{mec_host}/apps/{app_instance_id},更新DB status字段